INTERACTIVE SITUATION MODELS FOR SYSTEMS DEVELOPMENT

Meurig Beynon, Richard Cartwright, Jaratsri Rungrattanaubol, Pi-Hwa Sun

Department of Computer Science, University of Warwick, Coventry CV4 7AL

ABSTRACT

Devising principles for systems representation and analysis that can cope with the complexity of the interactions between programmable components and human agents in modern computing applications is a challenging and fundamental problem. Understanding the role of human and inanimate components within a reactive system, for instance, involves not only input-output transformations, but also communication and stimulus-response issues. This paper proposes novel computer-based *interactive situation models* to assist systems development. Such models provide an environment within which the human interpreter can explore the relationships between observables and the patterns of behaviour associated with a system component with particular reference to its external real-world semantics. They are constructed using principles based upon observation, agency and dependency ("Empirical Modelling") that have been developed at the University of Warwick. This paper describes and illustrates the characteristics of interactive situation models in detail, and concludes with a brief discussion of their potential significance for systems development.

1. INTRODUCTION

As modern computing applications continue to become more sophisticated, the interactions between computers, human agents, interfacing devices and other electronic devices such as sensors and actuators are ever more significant. Mathematical abstractions such as predicates and functional relationships can be effective in the context of sequential interaction between the user and the computer, but are insufficient as mental representations for state-of-the-art computer applications. The need for new concepts and abstractions to address modern software development has been discussed at length by F P Brooks [15] and Harel [18]. Brooks has emphasised the particular need for principles that can bring conceptual integrity to large-scale software development. Reactive systems, characterised (cf. Pnueli [26]) by embedded components, and concurrent real-time interaction, pose particular challenges. In such systems, it is not possible to abstract the role of a program from its context. Even very simple functional components can generate complex behaviour through concurrent interaction.

This paper discusses and illustrates how Empirical Modelling can be used to construct interactive situation models to assist reactive system development (cf [5,10,12]). Empirical Modelling (EM) is a collection of principles, techniques and tools that have been developed at the University of Warwick (see the EM web site[1] for additional information). An interactive situation model (ISM) is a computer-based environment constructed through situated[2] modelling activity. Unlike a closed-world computer model with a fixed interface, an ISM is always open to elaboration and unconstrained

---

[1] `http://www.dcs.warwick.ac.uk/modelling`

[2] The use of the term "situated" reflects the significance of both the social and cultural context (cf. [29]), and the specific physical environment in which the modelling is conceived.

exploratory interaction. States within the ISM metaphorically represent pertinent situations from the application domain, and possible transitions between states are explicitly constructed so as to be consistent with the developer's construal of a system in terms of agents, observables and dependencies [8]. An intimate relationship between the structure of the ISM and the developer's evolving knowledge of the domain and conception of the system is established in this way. The ISM is elaborated by the developer in an open-ended manner as understanding of the domain and the system is acquired. Interaction with the ISM serves to give empirical insight into the domain and the developing system. ISMs give particular support to the early stages of system development, but can in principle be useful at any stage.

The remainder of the paper is organised in three main sections. Section 2 motivates the distinctive style of state-based modelling associated with EM. Sections 3 and 4 discuss the principles behind applying EM to the construction of interactive situation models. These are illustrated with reference to an ISM for a digital watch. Further issues and future research directions are examined in the concluding section.

2. CONCEPTUAL REPRESENTATIONS FOR INFORMATION SYSTEMS

2.1. Motivation

Classical computer programming (cf Harel's one-person programming [18]) relies upon absolute guarantees about the context for human-machine interaction. These guarantees are gained through prior empirical study of the kind that is involved in developing hardware and testing a protocol for computer use. Even when fault-tolerance is considered, the same programming paradigm is typically applied. The empirical study is first extended to identify likely scenarios for human and machine error. The program (and system) is then developed to take account of these. In effect, classical programming is concerned with interaction in what has been empirically identified (for example, in the process of hardware design and development) to be a reliable and predictable framework. In such a framework, mathematical abstractions can be exploited relatively easily and directly.

In developing a reactive system, it is no longer possible to make a sharp separation between empirical analysis of a system and the process of programming the components. (The term *programming* is here used broadly to refer to all activities that involve specifying protocols for interaction for the state-changing agents within the system. This includes specifying operating rules for human agents, and providing settings for sensors and actuators, as well as conventional software development.) Experiential and empirical issues assume much greater importance: the correct functioning of the system relies on physical characteristics such as timing, matters of human perception and skill, speeds of processing and environmental factors. In this context, the developer has to act as an engineer and experimental scientist. This paper is concerned with how, through the use of ISMs, the computer can help to provide richer conceptual representations to support the developer in this role.

An ISM is conceived as a counterpart for system comprehension to Pennington's situation model for program comprehension [25]. Pennington's situation model was introduced by analogy with Kintsch and van Dijk's theory of text comprehension [22]. Both these models are text documents, rather than interactive and computer-based. There are several potential advantages to introducing an interactive situation model:

The role of interactive simulation in system development: In complex interactions within a reactive system, communication of state is very significant (cf. Deutsch's concern for stimulus-response patterns between agents [16]). It is helpful to be able to study such communications interactively, as practical tools for debugging have shown. On a related theme, Harel [18] contends that visual formalisms are needed to apprehend the semantics of complex systems.

The importance of empirical elements in an engineering context: Reliable knowledge about the interactions between agents in a system is an essential prerequisite to programming its components. An engineer wishing to explain a product with a view to maintenance or modification typically has to make use of artefacts to share the knowledge that informed the product development and the experimental activity that led to particular design decisions and structural features.

The pragmatic nature of program comprehension: Research by Good and Brna [17] has shown that - even for relatively simple programs - program comprehension can have many different complementary interpretations. An interactive situation model that can be subjective and is open to exploration, extension and revision is better suited as a program comprehension model than a document such as a program summary. In any complex system, there are typically many different perspectives on a design, and a variety of demands for intelligible representations.

2.2. Distinctive Characteristics of Interactive Situation Models

The principal characteristics of ISMs have been summarised in the introduction. Throughout this paper, the ISM concept will be discussed and illustrated with reference to modelling a digital watch. By way of illustration, Figure 1 depicts an ISM that has been constructed from the perspective of a proficient digital watch user. Before looking in greater detail at the principles used to construct this ISM, it is important to explain its distinctive character in more abstract terms. The nature of the domain knowledge that informs the ISM is particularly significant (as discussed in [14] and [24], closely related connections between knowledge and experience have been explored by William James e.g. in [21]):

- an ISM does not necessarily represent goal-oriented knowledge

  The idea behind the ISM is best understood by imagining a person (*the modeller*) observing and interacting with a real digital watch. In broad terms, the purpose of the ISM is to provide computer support for the modeller's conception of the watch. For the modeller, the ISM captures knowledge about the watch in a metaphorical fashion. In this context, "knowledge about the watch" refers broadly to anything the modeller can observe about the watch through interaction with it. This can range from naïve and ephemeral observations about its current state ("the face of the watch is shining in the sun"), to sophisticated expectations about its responses that have been acquired over a long period of interaction ("the watch keeps good time"). In particular, it need not be oriented towards a particular goal for interaction with the watch.

- an ISM represents subjective knowledge

  Unlike a conventional mathematical model, such as a finite state machine, the ISM is primarily subjective in character. Though a particular ISM may be useful in communication between one modeller and another, it reflects the perception and experience of a particular person of a particular watch in a particular context. To

appreciate the scope for subjectivity, contrast the likely conceptions of a digital watch for a child, an electronic engineer, a well-educated user, a naïve user, and a mediæval mystic.

- an ISM represents knowledge about interaction in particular contexts

  Constructing a computer model of a system typically means acquiring comprehensive objective knowledge about its behaviour in relation to a specified goal. An ISM is necessarily different in character. A child's naïve conception of a watch may exclude objective knowledge about its intended functionality and modes of user interaction. The knowledge captured in an ISM accordingly has a different ontological status, and requires another mode of representation. In an ISM, the computer is used to generate a direct metaphorical representation of a particular system state, and to animate the modeller's expectations about what latent state-changing actions are available, and what immediate consequences result.

The distinction between an ISM and a formal model of the behaviour of a digital watch can be elaborated with reference to Figure 1. Outside the context of the ISM, the statechart in Figure 1 specifies the watch as a finite state machine at an appropriate level of abstraction. For instance, each mode of display for the watch is represented by a node in the statechart, and each transition between display modes is represented by an edge that is coloured according to which input button is pressed. The statechart is then viewed as representing reliable knowledge about how the watch will react to input. From the modeller's perspective, specification of this nature is appropriate because the associated pattern of behaviour of the watch has been empirically established beyond any room for doubt.

The ISM has a different interpretation. Its semantics is defined with reference to both the mind of the modeller and the behaviour of the computer model. The modeller has in mind an explanatory account of how a digital watch operates, expressed in terms of observables, agency and dependency. The computer model metaphorically represents a particular state of the watch, together with latent state transitions that reflect what the modeller expects to happen in response to a particular action. There is an intimate relationship between the modeller's explanatory account and the construction and structure of the ISM, as is explained in more detail in later sections.

Within the ISM, the statechart serves as a metaphorical representation of the current mode of display of the watch. It also represents the modeller's expectations about how the mode of display will be affected by button inputs. In this context, the statechart does not specify that the behaviour of the ISM conforms absolutely to the pattern of a finite state machine. There is no circumscribed pattern of operational behaviour associated with the ISM. All the changes of state within the ISM are at the discretion of the modeller, though they can be delegated to automatic agents if desired, and can be executed without interference in order to simulate particular observed behaviours.

The essential idea behind using an ISM is to act within an empirical framework in which no absolute knowledge about system response is presumed. A digital watch may not always respond reliably and predictably to button presses, and the modeller retains the discretion to act as a rogue agent to simulate malfunction. The most intriguing aspect of this power to act outside the scope of any preconceived interaction is that the modeller can in principle simulate any behaviour, however outrageously unexpected. For instance, should a mediæval mystic presume that a particular

incantation can transform the digital watch into a four-legged beetle, this behaviour can be metaphorically realised.

As will be illustrated later, the close relationship between the modeller's causal account of the system and the structure of the ISM is especially significant when putting such open-ended behaviours into perspective. An ISM is based on a good explanatory model if system behaviours commonly observed in practice are easily derived from it, and if plausible singular behaviours can be readily simulated. In effect, an ISM implicitly attaches some measure of plausibility to the potential responses of the system.

This is consistent with the way in which different people assess the credibility of claims about observed behaviour of digital watches. From the perspective of an educated contemporary mind, it is inconceivable for a digital watch to be transformed into a beetle, since all the empirical evidence used to construe the behaviour of the watch has to be overwritten. In the broad context of ISMs for reactive systems modelling, there is still a place for modelling even the most unlikely transformations. For a robot trained to discriminate between objects, or for the squeamish human agent, there may be dangers in mistaking the image of a digital watch for that of a large beetle.

3. EMPIRICAL MODELLING PRINCIPLES AND SYSTEM COMPREHENSION

3.1. Observation and Explanation in Empirical Modelling

Constructing models to represent the interactions between the agents in a reactive system has been a central focus for research in the EM project for some time. In this context, an agent refers broadly to any component of the system that can be responsible for changes of state: for example, a computer, a sensory device, an actuator, a clock, a switch or a human agent. The digital watch and statechart simulation in Figure 1 is one example of a model constructed using EM principles (cf [13]). Other examples described in previous papers [5,8,12] include a vehicle cruise controller, a billiards game simulation, and a railway simulation. The term "interactive situation model" has been introduced subsequently as a synonym for "model generated using EM principles and tools".

EM supplies a framework for concurrent systems conception in which the basic abstractions are observables, dependencies between observables, and agents that act through changing observables and dependencies. Simple informal definitions for these concepts are:

- an **observable** is some feature of a system to which a value or status can be attributed in a system state. There are empirical procedures and conventions associated with identifying a particular observable and assigning its value. Not all the observables associated with a system need be present in a particular system state.
- an **agent** is a family of observables whose presence and absence in system states is correlated in time, that is typically deemed to be responsible for particular changes to observables within the system. All changes to the values of observables within a system are typically construed as due to actions on the part of agents.
- a **dependency** is a relationship between observables that pertains in the view of a particular agent. It expresses the empirically established fact that when the value

of a particular observable x is changed, other observables (the dependants of x) are of necessity changed in a predictable manner as if in one and the same action. The changes to the values of x and its dependants are indivisible in the view of the agent. That is: no action or observation on the part of the agent can take place in a context in which x has changed, but the dependants of x have yet to be changed.

The identification of observables, dependencies and agents is arguably a process that underlies all system construction, whatever the nature of the programming activity and paradigm is used, even if this process is not explicitly addressed. This paper argues for an intimate connection between comprehending a system and interpreting it in terms of these fundamental abstractions.

As explained in section 2, an ISM is a subjective model to be interpreted with reference to the modeller's explanatory account of the system. In effect, there are two complementary aspects to the situated modelling process:

- describing the abstract explanatory account of the situation that the modeller has in mind;
- constructing an ISM to imitate the observed responses to experimental and exploratory interaction.

The description of the explanatory account involves agent-oriented analysis based on a special-purpose notation called LSD. This analysis generates a text document: an *LSD account* of the situation. The LSD account captures the modeller's conception of the situation with reference to what observables and dependencies are significant, what agents are present, and how observables are classified with respect to these agents. A number of tools have been developed (and are still under development) for the construction of the ISM: the one that has been most extensively used, and that was used to construct the ISM for the digital watch in Figure 1 is the eden interpreter (also known as *tkeden*). Brief details of LSD and tkeden will be given here: for other details consult [5,6,14] and the EM web site.

3.2. An LSD account of the Digital Watch

The principles of LSD specification will be sketched with reference to a particular example. Listing 1 is an outline of the LSD account for the digital watch that is associated with Figure 1. (This account is deliberately modelled on the analysis of a digital watch that informs Harel's statechart (cf [19,13]).) The account can be read as expressing the modeller's belief about how the watch operates. The physical watch is represented in each state by instantiations of agents specified in the LSD account. The watch agent can be regarded as representing the watch and the power agent its power supply.

Within the specification of each agent the identifiers refer to observables that are significant for the agent. Some of these observables are associated with the agent itself, and do not exist when the agent is absent. Such an observable is classified as a **state** for an agent. For instance, `power_s` represents the voltage that is supplied to the watch: this is not a meaningful observable in the absence of the battery, and can otherwise take on three values, according to whether its charge is strong, weak or zero.

Within an agent, the special observable `LIVE` indicates whether the agent is present or absent. A **derivate** for an agent declares a dependency between observables in the view of that agent, so that for example:

```
LIVE = (power_s >= 1)
```

indicates that the watch is operative whilst the power is non-zero.

The specifications of the watch and main agents include subagents. The pattern of instantiation of these subagents determines how they contribute to the functionality of the agent in which they appear. In the case of the watch agent, the subagents are instantiated whilst it is instantiated, as is explicitly indicated by the LIVE derivates for the subagents `main` and `alarm_st`. In the case of the `displays` agent, precisely one of the subagents is instantiated whilst `displays` is active, according to which mode of display (as determined by the values of the observable `displays_s`) currently pertains. These two ways of composing families of subagents are closely related to the concepts of *orthogonality* and *depth* in a statechart (cf. [13] and [19]).

The behaviour of agents within a system is construed as determined by a protocol for action. Each action in the protocol is expressed by a possible stimulus-response pattern. For example, for the `alarm_st` agent, the interpretation of the action

```
displays_s == A & alarm_s == D & !d → alarm_s = E
```

is: when the alarm time is being displayed and the alarm is currently disabled, a possible response to pressing button `d` is to enable the alarm. An observable that is deemed to influence the behaviour of an agent (such as `displays_s` or `alarm_s` in this context) is classified as an **oracle**. An observable that can be redefined by an agent in the course of an action is classified as a **handle**.

The LSD account should not be mistaken for a formal model of system behaviour. The operational inadequacy of the LSD account is apparent in many aspects:

- The actions within an agent protocol are not obligations to act according to a rigid stimulus-response pattern. If an LSD agent to represent the user of the watch is introduced, their protocol will feature an action to silence the alarm. There can be no guarantee about how soon a user will perform this action, or even *whether* they will.
- The interpretation of agent actions is subject to any dependency relationships that pertain in the view of the observer of the system. By way of illustration, an agent that is responsible for ringing the alarm may have as an observable a derivate

  ```
  ring_alarm = (alarm_s == E) & (time == alarm_time)
  ```

  The action of enabling the alarm (`alarm_s = E`) may then be deemed to affect the observable `ring_alarm` indivisibly via a dependency.
- As the use of the term 'oracle' suggests, the knowledge that an agent has of an observable is subject to all manner of qualifications concerning the mode and accuracy of observation. As a simple example, the communication between the button interface and the display can vary from function to function. A button press may be registered as a signal for exactly one state-change, or be interpreted as one or more signals according to its duration. In contrast, the light on the display typically shows whilst a button is depressed.

- The synchronisation of agent actions in concurrent interaction is ambiguous. For instance, the effect of pushing two buttons simultaneously is unspecified. The protocols for the `displays` and `disp_alarm` agents potentially give conflicting responses to changes to the boolean `2-min`, which registers 2 minutes since the last input.

These unresolved issues highlight the fact that LSD agents are not conceived as closed-world abstractions. The context for their interaction, and the viewpoint of an objective external observer are conspicuously absent. The identifiers in the specification are intended to reference agents and observables in the situation that informs the situated modelling activity. As discussed in section 1, the stance of the modeller in framing an explanatory account and in constructing an ISM is an empirical one. The LSD account reflects what the modeller construes to be causal connections between changes to system state on the basis of empirical evidence. The ISM is constructed as a metaphorical representation for the system that is consistent with the modeller's construal. The purpose of the ISM is to enable the modeller to assess the quality of their explanatory account by exploring the implications of particular patterns of agency and interaction.

4. THE ROLE OF INTERACTIVE SITUATION MODELS IN SYSTEM COMPREHENSION

4.1. Constructing an Interactive Situation Model

The distinctive qualities of ISMs outlined in the introduction stem from the way in which states and transitions are represented in EM. The key idea is that the ISM is a model of the relationship between a situation and an observer, so that a modification of the model can either reflect a development in the situation ("an external change of state") or a new realisation on the part of the observer ("an internal change of mind"). The character of an ISM in this respect is similar to a spreadsheet, where the semantics of possible actions that might be performed by the user include:

- the update of a cell to reflect a change in the external situation;
- the correction of a cell value to make the spreadsheet consistent with the current situation;
- the modification of a defining formula to reflect a new insight into the situation, either to correct or to refine;
- the introduction of a new cell and defining formula to record additional information about the situation.

In an ISM, the situation is represented by a script of definitions (a *definitive script*) resembling the defining formulæ in a spreadsheet. In this context, the term *definition* is used in a technical sense to refer to an expression of the form $q = f(x, y, z, \ldots, t)$, where q, x, y, z, …, t are variables in the script. Each variable in the script represents an observable in the situation. Each definition represents a dependency between observables. The current values of variables in the script should conform to the current observations made of the corresponding observables in the situation. These values metaphorically represent the current situation. Transitions within the ISM are associated with redefinitions of variables (possibly performed in parallel). Changes to

values of variables are propagated in an indivisible fashion via the dependencies associated with definitions.

The principal tool that has so far been developed for EM, the tkeden interpreter, maintains dependencies in definitive scripts in which variables can assume many different types. An important aspect of tkeden is that scripts can be formulated using variables whose values are geometric in nature: these include points, lines, and windows (cf. [8]). This is illustrated in Figure 1, where each line of the LCD display is functionally dependent on the current time, as recorded internally as a scalar value (viz. "the number of seconds elapsed since a reference time").

In appreciating the significance of the ISM, it is helpful to imagine an idealised state-changing regime in which the modeller has discretionary control over every redefinition that occurs. Conceptually, the changes of state in the ISM should reflect as far as possible the expectations of the modeller. The dependencies in the model should account for all changes that are indivisible and cannot be interrupted (at least from the modeller's perspective). All other changes of state observed in the situation should be consistent with the modeller's explanatory account of the system behaviour, in the sense that they are plausibly attributable to the actions of agents represented in the LSD account. The state of the ISM can then be updated by the modeller, who performs the appropriate family of parallel redefinitions.

This state-changing strategy is idealised both in practice and in principle:

- **in practice**: It is quite impractical for the modeller to monitor every possible redefinition in the model. It is essential to introduce mechanisms for agent actions to be performed autonomously. In Figure 1, for instance, the current time is automatically incremented, and all relevant dependencies maintained, every second. This is achieved in tkeden by introducing triggered redefinitions. This mechanism does not prevent the modeller from interacting with the model, but – in some contexts – it can introduce an inappropriate element of oblivious execution.
- **in principle**: A modeller's explanatory account cannot be so comprehensive as to deal with every scenario for interaction that can be invoked in a situation. If there is no constraint on the scenarios the modeller can invoke, then the frame problem applies [27]. Even when the mode of observation and the pattern of interaction is restricted, it is still possible in general for there to be a conflict between actions (as between the `disp_upalarm` and `displays` agents when the display is timed out)

There are two aspects to the above idealisation. The practical limitations of our tools motivate new developments of two kinds: the design and implementation of new architectures for EM (cf [2,4]), and the enhancement and refinement of tkeden. In the context of this paper, the most relevant developments are the recent introduction of a distributed tkeden architecture, based on a client-server model, and the development of techniques for more effective high-level analysis of tkeden models. By using these innovations in combination, it will be possible to interpret different parts or views of a system on different client workstations, and to provide an appropriate environment for integrating and monitoring the interaction between these on the server.

The fact that *even in principle* there are problems in developing an explanatory account to reconcile interaction with the ISM and interaction with the situation is the essential point of EM. This reconciliation is an ideal to which the modeller aspires, but cannot in general attain. A very high degree of consistency between 'what is observed'

and 'what is consistent with an explanatory account' is characteristic of *closed-world* models. As conventional computer-based modelling illustrates, it is possible to construct excellent closed-world models in certain contexts. One of the main reasons for developing ISMs is to support the processes that precede the identification of such models. Another is to provide computer assistance in contexts where there is no realistic possibility of constructing a closed-world model. ISM construction is essentially concerned with the creative tension between 'what can be observed' and 'what can be explained'.

4.2. An ISM for a Digital Watch

The above discussion of ISMs can be illustrated with reference to the ISM for a digital watch in Figure 1. The history and construction of this ISM is also examined from another perspective in [13].

The visual elements in the ISM represent current values of the observables that establish a particular state of the watch. In constructing the ISM, the modeller works in situated mode, in (or more realistically *as if in*) the presence of a real watch. Relevant observables for the ISM include all characteristics of the actual watch that can be reliably identified, possibly over an extended period of interaction with the watch, perhaps using special instruments and experimental techniques. Examples of such observables include: the appearance of the watch, the digits that appear on the LCD display, the layout of the buttons, the presence or absence of a power supply, the voltage currently being supplied to the watch and the current mode of display.

It is clear that the particular choice of observables in Figure 1 is to some extent complementary to the LSD account in Listing 1. There is nothing to constrain the choice of observables to suit a particular explanatory goal, however, and there is scope for purely speculative and exploratory extension of the ISM. For instance, greater realism in the visual representation of the digital watch would not entail re-engineering the existing ISM in Figure 1. Adding new observables to the model can be viewed as recruiting more observers in the spirit of subject-oriented programming. Just how seamlessly this introduction can be made is constrained only by how far the conception of the watch in the minds of these new observers is consistent with that of the original modeller.

As Figure 1 illustrates, the metaphorical representation of observables of the watch can be abstract rather than realistic. The depiction of the watch resembles an actual watch only in the crude sense that (say) it is approximately rectangular in shape, has an LCD display, and has a button located at each corner. The use of numbering and colouring metaphorically expresses the fact that the user can distinguish the buttons. It also suggests that the identity of the individual buttons is explicit, but in practice a simple physical skill may have to be learned to make this identification. The presence of a synchronised analogue clock in Figure 1 conveys the idea that the numerical display is to be interpreted as a time. The statechart in Figure 1 is the most sophisticated metaphorical device in the representation. Highlighting a particular region within the statechart is a means of expressing the significance of the current display. Notice that this addresses an aspect of the state of the actual watch that the user cannot in general observe directly. For example: when the display mode is set to `chime`, the LCD shows the time at the next hour; when the mode is set to `alarm`, the LCD shows the time at which the alarm will go off. If these times coincide, the mode of the watch can be determined only with reference to past or future interaction.

The dependencies in the ISM of Figure 1 reflect several different types of agency. The primary purpose of these dependencies is to bind the visual elements and invisible internal state of the model together so to reflect interaction with the external observables they represent. Though the ISM incorporates a simple interface through which the modeller can simulate the button pressing behaviour of the user, it is the textual tkeden interface that properly reflects the modeller's discretion over the model. This interface gives scope to extend and modify all the definitions, functions and triggered actions interactively. Changing the current time affects the displays on the digital and analogue watches indivisibly. Moving the minute hand of the analogue clock moves the hour hand. Changing the mode of display indivisibly affects the highlighted state in the statechart and the watch display.

The significance of the ISM in explanatory terms is bound up with developing mechanisms that deal as fastidiously as our present tools allow with agency and communication between agents (contrast what Smith [28] characterises as 'promiscuous modelling'). This involves more than making a convincing simulation; it also means applying EM principles to the models of internal state. Reconciling the LSD account with the ISM is bound up with the process of realising the behaviour of the system from the proposed account of its constituent agents. For instance, Listing 1 indicates that the observable `displays_s` is dependent on `upalarm_s` whilst the agent `disp_alarm` is active, and reverts to an explicit value when the agent is dismissed.

The use of ISMs illustrated in Figure 1 and Listing 1 is biased towards what Hirschheim et al [20] classify as a *functionalist* paradigm for software development. When viewed from this perspective, EM can be seen as a modelling technique resembling existing specification methods such as ESTEREL [3] and application builders such as ACE [23]. The essential character of EM is better conveyed in relation to applications where some subjective element is involved (cf [11] and [14]).

For this purpose, observables for the digital watch must take account of the knowledge and perception of the human agent who interacts with it. Aspects of the appearance of a watch can be subjective, and may depend upon environmental factors, such as the ambient light. Some intelligence is needed to identify a digit rather than pattern of illuminated segments. Reading the current time involves knowing social conventions. Subtle ergonomic issues govern a user's perception of the buttons. Recognising the mode of display presumes sophisticated knowledge of the range of functions performed by the watch. Familiarity with the user manual is required to make the association between buttons and the functionality of the watch.

By way of illustration, the ISM in Figure 1 can be simply adapted to serve as a learning aid for a digital watch. Two models are run concurrently, in a tkeden client-server configuration. In the client model, the image of the statechart is suppressed. Communication between the models is set up in such a way that button input in the client is monitored by the client. This creates an environment in which a teacher can give instruction on the functionality of the watch. This may involve watching the way in which the learner's actions affect the statechart, intervening to assist the learner to restore the state of the watch, or possibly introducing the statechart incrementally into the client ISM as the learner acquires the relevant empirical knowledge about the functionality of the watch. The teacher might also attach agents to process the learner's responses, perhaps in relation to certain set goals.

As a final illustration of the support for agency that EM affords, meta-level methods of processing ISMs have also been explored. A tool to specify and animate statecharts interactively (using a GUI) has been developed in tkeden. The output from this tool is a tkeden model of a statechart that can be integrated with an existing ISM through introducing appropriate dependency relations between observables. Other recent work has addressed generic ways of transforming the text of an ISM to create a metamodel.

By including the transformed text through the tkeden interface, an agent that (e.g.) displays the tkeden identifier of a selected window can be introduced into any existing ISM. Meta-agents of this nature complement the inventory of definitions and the command history that are built-in features of tkeden.

5. CONCLUSION

This paper has focused on exposing and illustrating the essential character of interactive situation models. It has argued the case for ISMs as novel computer-based models whose interpretation requires a radical shift of perspective. Previous papers have considered potential applications for ISMs in connection with a wide range of system design and development issues. These include: requirements modelling [10], software construction [6], concurrent engineering [1] and program comprehension [7].

Some research has been carried out on semi-automatic translation from ISMs to conventional programs and simulations (cf [2]). Related work by Adzhiev and Richlinskey at the Moscow Engineering Physics Institute involves the direct application of EM principles to generate object-oriented programs semi-automatically. Such research points to a futuristic scenario in which an ISM is supplied with a system. This could be used as an extensible basis for system comprehension, as a way of linking requirements specification and validation, and as a knowledge base to support maintenance.

One of the open issues concerns the size of ISMs that can be constructed in practice. The representations for ISMs in the tkeden interpreter are relatively inefficient, but can deliver reasonable performance whilst maintaining a script of over a thousand definitions on a modest workstation. Scripts of this size have potentially enormous expressive power as state-transition models. Several developments promise improved capacity and efficiency (cf [2]). These include: the use of higher-order definitions to eliminate repetition of patterns of observables, dependency and agency; the distribution of ISMs across a client-server architecture; the development of new architectures in which dependency is maintained close to the machine code level, and optimised parallel algorithms for storing and updating dependencies.

What is clear is that EM prescribes no general *method* for system development. The process of trying to reconcile the responses of an ISM with an explanatory account is open-ended and cannot be guaranteed to lead to convergence. Progress in constructing an ISM relies upon experimental activity in which creativity and serendipity are intrinsically involved. To return to the themes of Brooks's seminal text [15], EM is no more a silver bullet for software technology than the so-called "scientific method" is a recipe for creating scientific theories. What EM offers is a framework of principles and tools that promises to deliver greater conceptual integrity across the entire development process. Observables, agency and dependency have their place in the subjective realm of the modeller's experience [11], can be used to represent the interaction between many designers [1], to account for the behaviour of

families of diverse components within a reactive system and provide the basis for new abstract computational models and architectures [2,4].

ACKNOWLEDGMENTS

We are grateful to the members of the Empirical Modelling Research Group, and to many undergraduate students whose project work has contributed to this paper. We are particularly indebted to Steve Russ and Dominic Gehring for their valuable critical and constructive interest in the agenda of this paper.

REFERENCES

1. V D Adzhiev, W M Beynon, A J Cartwright, Y P Yung *A Computational Model for Multi-Agent Interaction in Concurrent Engineering* Proc CEEDA'94, Bournemouth Univ., 1994, 227-232

2. J Allderidge, W M Beynon, R I Cartwright, Y P Yung *Enabling Technologies for Empirical Modelling in Graphics* CS-RR#329, University of Warwick 1997

3. G Berry, G Gonthier *The ESTEREL synchronous programming language, design, semantics, implementation* Science of Computer Programming 19(2), 1992, 87-152

4. W M Beynon, M D Slade, Y W Yung *Parallel Computation in Definitive Models* CONPAR 88, BCS Workshop Series CUP, 1989, 359-367

5. W M Beynon, I Bridge, Y P Yung *Agent-oriented Modelling for a Vehicle Cruise Control System* Proc ESDA'92, ASME PD-Vol.47-4, 1992, 159-165

6. W M Beynon, M T Norris, S B Russ, M D Slade, Y P Yung, Y W Yung *Software Construction using Definitions: an Illustrative Example* CS-RR#147, University of Warwick 1989

7. W M Beynon, Pi-Hwa Sun *Interactive Situation Models for Program Comprehension* Univ of Warwick 1998

8. WM Beynon *Agent-oriented Modelling and the Explanation of Behaviour* in Proc Int Workshop on Shape Modelling, Parallelism, Interactivity and Applications, Dept of Computer Software TR94-1-040, Univ of Aizu, Japan 1994, 54-63

9. W M Beynon, Y P Yung, A J Cartwright, P J *Horgan Scientific Visualisation: Experiments and Observations* Proc Eurographics Workshop on Visualization in Scientific Computing, 1992, 157-173

10. W M Beynon, S B Russ *Empirical Modelling for Requirements* CS-RR#277, University of Warwick 1994

11. WM Beynon *Modelling State in Mind and Machine* CS-RR#337, University of Warwick 1998

12. W M Beynon, M Farkas, Y P Yung *Agent-oriented Modelling for a Billiards Simulation* CS- RR#260, Univ of Warwick 1993

13. W M Beynon, R I Cartwright *Empirical Modelling for Cognitive Artefacts* IEE Colloquium, Dec 1995, Digest #95/231, 8/1-8/8

14. W M Beynon *Empirical Modelling for Educational Technology* Proc Cognitive Technology '97, IEEE 1997, 54-68

15. F P Brooks *The Mythical Man-Month Revisited* Addison-Wesley 1995

16. M S Deutsch *Focusing Real-Time Systems Analysis on User Operations IEEE Software* Sept 1988, 39-50

17. J Good, P Brna *Explaining Programs: when talking to your mother can make you look smarter* Proc PPIG-10 Annual Workshop, 61-69

18. D Harel *Biting the Silver Bullet: Towards a Brighter Future for System Development* IEEE Computer, Jan 1992, 8-20

19. D Harel *Algorithmics* Addison-Wesley 1987

20. R Herschheim, H K Klein, K Lyytinen *Information Systems Development and Data Modelling: Conceptual and Philosophical Foundations* CUP 1995

21. William James *Essays in Radical Empiricism* Bison Books 1996

22. W Kintsch, T van Dijk *Toward a model of text comprehension and production* Psychological Review, 85, 363-394

23. B Nardi *A Small Matter of Programming: Perspectives on End-User Computing* MIT Press 1993

24. P Naur *Knowing and the Mystique of Logic and Rules* Kluwer Academic Publishers 1995

25. N Pennington *Comprehension strategies in programming* Empirical Studies of Programmers: Second Workshop, New Jersey, Ablex Publishing Co. 1987, 100-113

26. A Pnueli *Applications of Temporal Logic to Specification and Verification: A Survey of Current Trends* Lecture Notes in Computer Science #224, Springer-Verlag, 1986, 510-584

27. Z Pylyshyn (ed) *The Robot's Dilemma: the Frame Problem in Artificial Intelligence* Norwood NJ, Ablex 1986

28. Brian Smith *Two Lessons in Logic* Computer Intelligence Vol 3, 1987, 214-218

29. L A Suchman *Plans and Situated Actions: the Problem of Human-Machine Communication* Learning in doing: Social, cognitive and computation perspectives, CUP 1987

Listing 1: Outline LSD account for the Digital Watch (cf. Figure 1)

```
agent power() { state power_s = battery_charge }                                    // three-valued 2,1,0
agent watch() {
    derivate   LIVE = (power_s >= 1)
    oracle     power_s
    agent main() {
        derivate   LIVE = LIVE_watch
        oracle     LIVE_watch , main_s, alarm_s
        state      main_s = D                                   // an integer -- 1:Displays, 2:Beep || displays
        handle     main_s
        protocol   (main_s == D) & (time == set_time) & alarm_s == E → main_s = B
        agent displays() {
            derivate   LIVE = LIVE_main
            oracle     LIVE_main
            state      displays_s = T    // 1:Time, 2:Update, 3:Date, 4:Stopwatch, 5:Alarm, 6:Update Alarm, 7:Chime
            handle     displays_s, update_s, upalarm_s
            protocol   displays_s == T & !c → update_s = 1,          // !c here means that button c has been pressed
                       displays_s == T & !d → displays_s = D,
                       displays_s == A & !c → upalarm_s = 1,
                       displays_s ≠ S & displays_s ≠ T & 2-min → displays_s = T          // display timeout
                       …
            agent disp_date() {
                derivate   LIVE = LIVE_displays & displays_s == D,
                           "watch_display = date as of clock()"
                oracle     LIVE_displays, displays_s
            }
            agent disp_time() { … }
            agent disp_upalarm() {
                derivate   LIVE = LIVE_displays & upalarm_s > 0,
                           displays_s = (upalarm_s % 4 == 0 ) ? A : UA
                           "watch_display = time as of alarm setting with right digit highlighted"
                oracle     LIVE_displays, upalarm_s
                state      upalarm_s = M                  // 1:Min, 2:TenMin, 3:Hr, 4 ≡ 0(mod 4):Alarm
                handle     upalarm_s, set_time
                protocol   !b or 2-min → upalarm_s = 0,
                           !c → upalarm++,
                           "event→ update set_time so as to increment highlighted digit"
            }
            …
        }                                                                    // end displays()
        agent beep() {
            derivate   LIVE = LIVE_main & main_s == B
            state      main_s
            protocol   beep_stop → main_s
        }
    }                                                                        // end main()
    agent alarm_st() {
        derivate   LIVE = LIVE_main
        oracle     LIVE_main, displays_s, alarm_s
        state      alarm_s = D, set_time = 00.00
        handle     alarm_s                                                   // 1:Disab, 2:Enab
        protocol   displays_s == A & alarm_s == D & !d → alarm_s = E
                   displays_s == A & alarm_s == E & !d → alarm_s = D
    }
    agent chime_st() { … }
    agent clock() { … }
    agent stopwatch() { … }                          // ... and several other agents, such as light() etc...
}
```

Figure 1: The digital watch Interactive Situation Model